(Click here for the web formatted version)

NaturalPoint Tracking Toolkits (Point Cloud and Rigid Body)

Users Manual version 1.0.033

Complete Table Of Contents

1. Introduction

1.1 Forward

Information in this Users Manual is subject to change without notice and does not represent a commitment on the part of NaturalPoint. The software described in this Users Manual is furnished under a license agreement and may be used only in accordance with the terms of said license agreement.

OptiTrack, Point Cloud toolkit, Rigid Body toolkit and *NaturalPoint* are trademarks of NaturalPoint Inc. Windows is a trademark of Microsoft. All other trademarks are property of their respective owners.

For providing VRPN, NaturalPoint would like to thank the NIH National Research Resource in Molecular Graphics and Microscopy at the University of North Carolina at Chapel Hill, supported by the NIH National Center for Research Resources and the NIH National Institute of Biomedical Imaging and Bioengineering.

1.2 About NaturalPoint

NaturalPoint Inc. is pleased to provide you with superior optical tracking products, we hope that you enjoy using your NaturalPoint Tracking Toolkits.

NaturalPoint
33872 SE Eastgate Circle
Corvallis, OR 97333
Telephone: 541-753-6645
Fax: 541-753-6689
www.naturalpoint.com

2. How to use the Manual

Quick Start

It is strongly recommended to read this manual before using the Tracking Toolkits optical tracking product. To begin using the product as soon as possible without reading the full manual then you can start quickly by following these instructions :

- ◊ Read the Installation section and follow the instructions described there, otherwise your Tracking Toolkits (Point Cloud and Rigid Body) may not work.
- ◊ Once the software is installed, connect the OptiTrack cameras to the USB ports of your computer.
- ◊ Activate the Point Cloud toolkit License
- ◊ Launch the OptiTrack Camera Calibration tool using the shortcut on your desktop.

Further Reading

The Tracking Toolkits (Point Cloud and Rigid Body) are powerful optical tracking solutions with a number of options designed to help you get the best performance. It is recommended that you read Section 4 ("Using Point Cloud Software"), Section 5 ("Using the Point Cloud API"), Section 6 ("Using Rigid Body Software") and Section 5 ("Using the Rigid Body API") to better understand how the product works.

3. Getting Started

3.1 Minimum System Requirements

- ◊ 3+ OptiTrack cameras
- ◊ Windows XP SP2 or Windows Vista
- ◊ .Net 3.0 Runtime
- ◊ VC2005 Runtime
- ◊ 1.5 GHz Processor
- ◊ 256 MB of RAM

◊ 20 MB of free hard disk space
◊ USB 2.0 Hi-Speed port

3.2 What's Included

1 Point Cloud or Rigid Body license card

1 Quickstart guide

1 Software CD

3.3 Hardware Compatibility

The Tracking Toolkit (Point Cloud and Rigid Body) software can only be used with OptiTrack FLEX:C120, OptiTrack FLEX:V100, and OptiTrack SLIM:V100 hardware. Older camera models are not compatible and will not work with this software.

3.4 Software and Hardware Installation

For best results it is suggested to install all software before you connect the OptiTrack cameras.

3.4.1 Software Installation
NOTE: Windows XP and Vista users must be logged in as an administrator. If you only have one user on your computer, you most likely already have administrator privileges.

1. Insert the included NaturalPoint software CD into your CD drive. Wait for the install program to start. If the install program does not start within a few minutes, open "My Computer" then double click on the CD drive icon, and double click again on the Setup file.

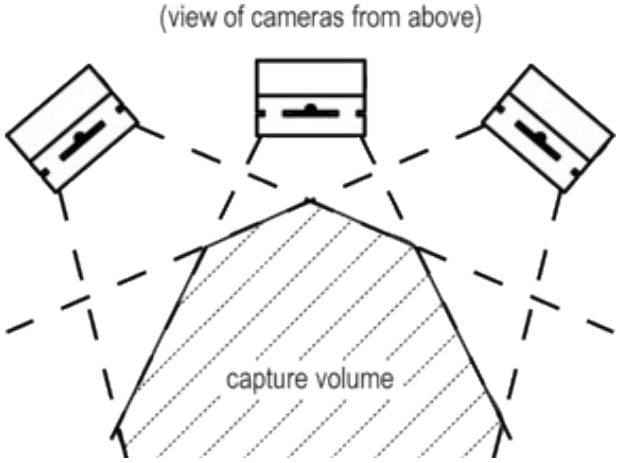
2. There are two software packages to install, the OptiTrack SDK and the Tracking Toolkits. Start with the OptiTrack SDK installer, once it has completed it will automatically start the Tracking Toolkits installer for you.
   Note: *The Tracking Toolkits will detect if you are missing required dependencies such as .Net 3.0 and install them for you.*

3. Follow the software installation instructions on the screen.
   Note: *Windows may display a warning message about the drivers not being signed. Click YES to accept the drivers, the drivers WILL NOT harm your system.*

4. OptiTrack SDK, Point Cloud and Rigid Body icons will appear on your desktop.

3.5 Camera Placement

In order to track markers, multiple OptiTrack cameras must be arranged to have overlapping fields of view. This will create an area called a *capture volume* in which tracking can occur. When possible, secure the cameras firmly in place. Whenever the cameras are moved it is necessary to recalibrate them.

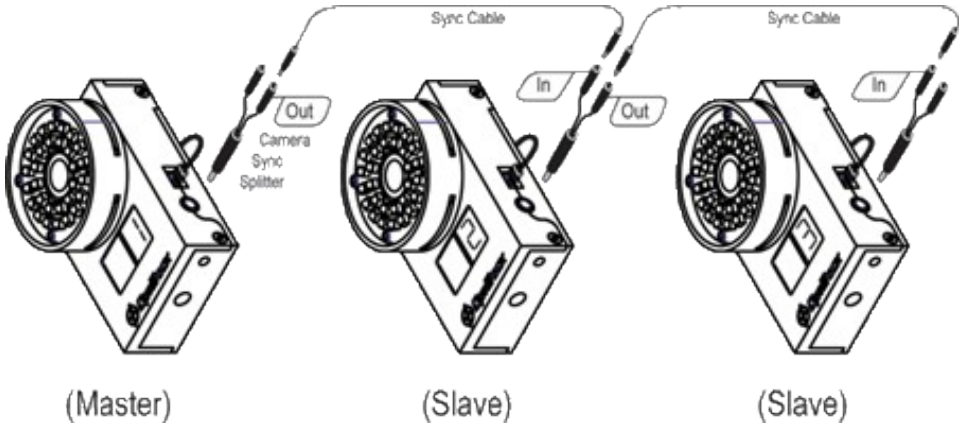


For best calibration and tracking results, avoid placing the cameras all in the same plane. Instead position the cameras at different angles.

3.6 Camera Synchronization

Multiple OptiTrack cameras can be linked together to synchronize their exposure timing. Camera synchronization allows for greater precision and more robust tracking. The cameras are connected in a chain using Sync Cables*. Once the cables are connected and the cameras turned on, they will begin synchronizing automatically.
** Sync Cables sold separately*

1. Note : **All cameras must be of the same model type, otherwise synchronization will not work and should not be attempted.**
2. Connect all cameras to the computer via USB.
3. Plug the Camera Sync Splitter into each camera.
4. Chain the cameras together by connecting each camera's "out" connector to the next camera's "in" connector.
5. Do not form a loop, the last camera in the chain should **not** be connected to the first camera.
6. Start cameras in the software, the cameras will automatically synchronize.



3.7 License Activation

Before the Point Cloud toolkit or Rigid Body toolkit can be used, you must first activate the included license (see License Card in section 3.2) for one of your OptiTrack cameras. Choose one of your cameras and use the serial number found on the back during the activation process. Activating the license enables you to use the software toolkit as long as the camera for which it was activated is connected to your computer.

There are two different methods for activating your license :

◊ Direct : Double click the OptiTrack License Activation tool shortcut on your desktop. Fill out the required details and then press the activate button. If the activation is successful a license will be generated and installed on your computer.

◊ Email : If your computer is behind a firewall which prevents the direct tool from working, you can activate your license on the OptiTrack website and have it emailed to you. Please visit www.optitrack.com/activate for more details.

License Activation tool :

Additional information can be found in our licensing FAQ online at http://www.naturalpoint.com/optitrack/support/activate/faq.html.

4. Using the Point Cloud Software

The Point Cloud toolkit is a powerful solution for your advanced tracking needs. It provides 3D tracking and calibration across multiple OptiTrack cameras in a compact software toolkit, all with an easy to use API. With the Point Cloud toolkit properly installed and calibrated, your applications will have easy access to robust and precise tracking data. The information provided below will help ensure that you get the best results from your OptiTrack cameras and Point Cloud software.

4.1 Getting Started

Before using the Point Cloud software you should make sure to have all of the software and hardware properly installed and licensed as described in Chapter 3. Once the installation is complete you can begin using the Point Cloud software by following these instructions :

1. Connect 3 or more cameras to the USB port of the computer and arrange them to set up a capture volume as described in Section 3.5 ("Camera Placement").

2. Run the calibration wizard to calibrate the cameras and test the capture volume.

3. Save the resulting calibration for later use in your own software.

4.2 Camera Calibration Tool

Before the Point Cloud software is able to track markers, it necessary to conduct a camera calibration first. The calibration allows you to fine-tune the OptiTrack camera settings, identifies the physical location of the cameras (*extrinsic parameters*) and the properties of their lenses (*intrinsic parameters*). The Calibration tool which guides the process also provides a way to review the results and capture sample data. Before starting the calibration you will need to have your cameras set up as described in Section 3.5 ("Camera Placement"). If any of the cameras are physically moved, it will be necessary to conduct a new calibration.

The following section describes how to use the calibration tool. To start the tool, click the "OptiTrack Point Cloud Calibration Tool" icon on your desktop.

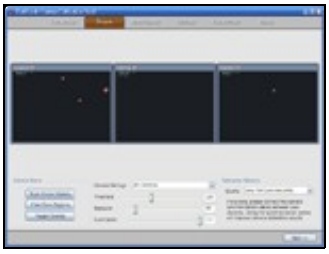
4.2.1 Licensing

The first step of the Camera Calibration verifies that a valid license and matching camera are present. It also checks to make sure that 3 or more cameras are connected to the computer. It is not possible to conduct the calibration and track with less than three cameras.

If everything is ready, press the "Begin" button to proceed to the next step.

4.2.2 Camera Preview

The Camera Preview step displays the scene contents from the perspective of the cameras. The purpose of this step is to remove all unwanted light picked up by the cameras. This can be done by removing physical items, turning off lights, covering windows, adjusting camera settings or using the blocking tools. Please be sure your ground plane and wand are not in the capture volume during this process.



Camera Views : Moving the mouse over an individual camera view will temporarily switch the camera into greyscale mode and display the actual capture volume from that camera's perspective. A camera view can be zoomed to and from full screen by double clicking in its border area.

Camera Settings

The OptiTrack cameras have settings which can be optimized to help reduce unwanted light in the scene. When adjusting the settings it is possible to change them for individual cameras, or all at the same time using the "Camera Settings" dropdown selector.

- ◊ **Threshold** : This scrollbar adjusts how bright a marker has to be before it will be identified by the cameras. Using a larger value such as "180" will help reduce the number of unwanted items in the

scene. Using too high of a value (over "230") can degrade the precision of the detected markers.

◊ **Exposure** : This scrollbar adjusts the camera's shutter timer which determines the amount of light a camera lets in during each frame. Using a smaller value such as "55" will help reduce the number of unwanted items detected in the scene.

◊ **Illumination** : This scrollbar adjusts the brightness of the illumination provided by the OptiTrack camera's on board IR LEDs.

For best performance when using OptiTrack FLEX:V100 cameras, use the IR strobe mode. This mode is activated by setting the illumination value to "15". When strobe mode is active there is no benefit to using an exposure value larger than "100" since the LEDs cease to emit light after that time. To adjust the brightness of the LEDs in strobe mode, modify the exposure setting within the range of "1" through "100" (smaller values will result in less light).

Blocking tools
The software can be instructed to ignore detected light (*markers, light sources, etc*) in the field of view of a camera by using blocking regions. Blocking regions can be added by manually drawing boxes on the camera previews using the mouse or automatically using the "Block Visible Markers" button. To remove all blocking regions press the "Clear Block Regions" button.

Calibration Options
The quality of the calibration results can be improved taking more time to complete. Typically the faster settings will be sufficient. Use the "Quality" dropdown selector to choose the desired calibration level.

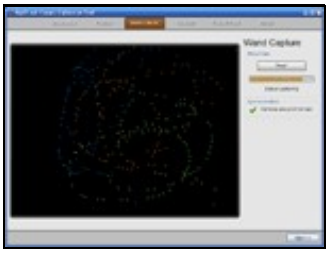
Once the scene is clear and has all markers either removed or blocked, press the "Next" button to proceed.

4.2.3 Wand Capture

The purpose of the Wand Capture step is to collect a set of data from the cameras which will be used to calculate their physical position and lens characteristics. A wand (*reflective marker with a handle*) should be moved through the capture volume in a three dimensional pattern at a moderate speed while the software records it. As the wand is moved through the scene it will be represented on the screen as a circle with a different color for each camera. When a sample point is taken it will show up on the screen in a darker color.

Special attention should be paid to obtaining good coverage for each camera. The recorded sample points can be used as a visual indicator and guide of wand coverage. Moving the wand in a circular, three dimensional pattern can also help in obtaining good results. Also try to wave the wand away and toward the cameras often to produce as much depth motion as possible.

It is important that the wand is the only detected marker in the scene for every camera. Additional markers can introduce errors and reduce the quality of the calibration. The software will warn the user and attempt to ignore additional markers, however if additional markers are present it is recommended to return to the previous "Camera Preview" step and remove them.



Wand Data
When you are ready to start the wand data capture process, press the Start button. As the data is collected the progress bar will be updated to indicate how much time remains.

Synchronization
The synchronization indicator provides information about how well the OptiTrack cameras are synchronized. Using hardware camera synchronization can greatly improve the performance of the calibration and 3D tracking. Please see Section 3.5 ("Camera Synchronization") for more information.

Calibration Options
Customizable settings are provided to help optimize the calibration process for the size and arrangement of specific capture volumes. Adjusting the values to those which best match the capture volume will help improve tracking performance.

Wanding Data
Wanding data can be saved in CSV format for later use and reloaded using the Load and Save buttons.

Once the wand data has been collected, press the "Next" button to proceed.

4.2.4 Calculate Calibration

Once a set of wand data has been captured in the previous step, it needs to be processed. The length of time taken for the calculation depends on the calibration quality level selected in the "Camera Preview" step. To start the calculation, press the "Start Calculation" button.

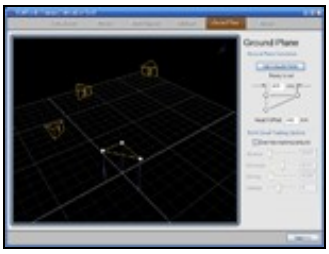
When the calculation is complete it will stop scrolling and display information about the level of quality obtained. The lower the numbers displayed for the Standard and Mean Error, the better. Typically it is desirable to have the Standard error below "0.5" and the Mean Error below "0.5". If the result of the calculation is unsatisfactory, consider returning to the "Wand Capture" stage to collect a new set of data.

Once a good calibration has been obtained, press the "Next" button to proceed.

4.2.5 Ground Plane

The Ground Plane step allows you to identify the level of the physical floor (ground), set the scale of the coordinate system and how the axes of the 3D world are oriented. The cameras will be displayed in 3D using the positions calculated during the calibration step. The software will also reconstruct and display 3D coordinate data for markers detected in the scene. Until the ground plane is set, the position of the cameras and markers may not appear in the expected positions.

The Ground Plane is a 3-4-5 triangle with reflective markers in the three corners of the triangle. When the ground plane is placed within the capture volume, its three markers should show up in the 3D view with lines between them. When the triangle is detected and there are only 3 markers in the scene, press the "Set Ground Plane" button. After pressing the button, the cameras and markers will be updated to their new positions.



3D View

◊ **Rotate View** : The 3D view can be rotated by pressing down the left mouse button and moving the mouse.

◊ **Translate View** : The 3D view can be translated by pressing down the right mouse button and moving the mouse.

◊ **Translate View** : The 3D view can be zoomed in or out by using the mouse-wheel or by pressing down the left and right mouse buttons and moving the mouse.

Ground Plane Calculation

◊ **Set Ground Plane** : Press this button to update the calibration when the Ground Plane is in the desired position. It is only possible to press the button when 3 markers are in the scene, additional markers will cause the button to be disabled.

◊ **Ground Plane Size** : Enter the size, in millimeters, of the "4" side of the Ground Plane 3-4-5 triangle. The "4" side is the Z axis. The default size is 300mm, 400mm, 500mm. It is important that this number is set accurately since it is used to scale the coordinate system for all tracking data. This number should be entered before pressing the "Set Ground Plane" button.

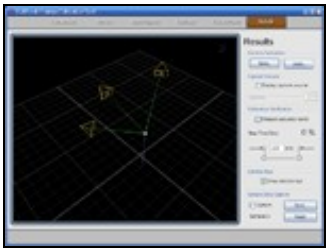
◊ **Height Offset** : If the Ground Plane is not physically located on the exact floor (ground), then a height offset can be entered in millimeters to compensate for the difference. This number should be entered before pressing the "Set Ground Plane" button.

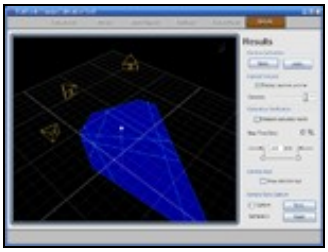
Point Cloud Tracking Options
If needed, the parameters which control the reconstruction of 3D markers from 2D position data can be fine-tuned manually. Typically the default values should provide good results.

Once the Ground Plane has been set, proceed to the next step by pressing the "Next" button.

4.2.6 Results

The calibration process is now complete. This last step provides a chance to review the results of the calibration and store them for later use. A number of tools are provided for visualizing the data and measuring the quality. If there is a custom application which will be making use of the Point Cloud COM object, it will need to load the calibration file saved at this stage in order to perform any tracking. It is also possible to load existing calibrations for inspection and use.





Camera Calibration

Use the Save and Load buttons to store calibrations for later use or retrieve previous calibrations.

Capture Volume

This tool provides a way to visualize the physical area of the capture volume to determine where there will be good tracking coverage. It can be adjusted to display the coverage area for all cameras, or to just show a subset where multiple cameras overlap.

Calibration Verification

It is possible to measure the quality of the calibration by using a wand with two markers at a known distance apart (an *accuracy wand*). When the "Measure accuracy wand" box is checked, the wand can be moved through the capture volume and a real-time estimation of the error for the wands position will be displayed. The distance of the points on the wand should be entered in millimeters.

Camera Rays

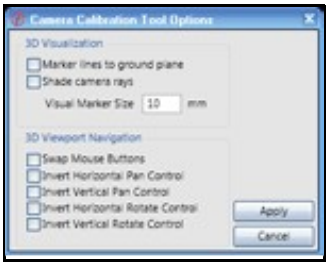
The Camera Rays tool can be used to determine which cameras are able to see which markers. It does this by drawing a line between every marker and the cameras which are detecting it.

Sample Data Capture

3D marker tracking data can be exported in CSV file format from this area by recording a capture sample and saving it. Use the button with the red circle to start recording and the button with the black square to stop. The "Save" button allows you to save the recorded data to a file. The exported data is formatted with the first column indicating the number of markers, the second column with a timestamp, and then 3 additional columns (x,y,z) per marker.

4.2.7 Options

Several aspects of the Calibration tool interface can be customized to better suit your needs. The settings which control customization are found in the Options window, which can be launched by "Options" button in the lower-left corner of the Calibration tool. Changes to these settings will be saved when the Calibration tool is closed, and restored when it is started again.

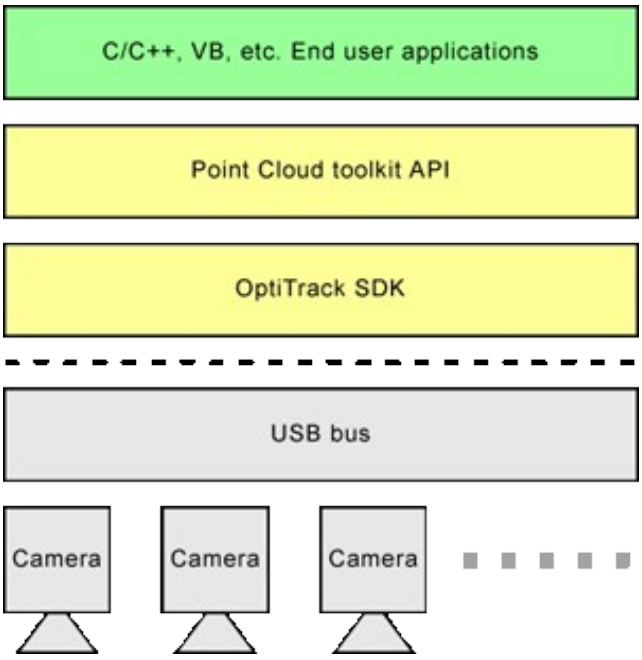


5. Using the Point Cloud API

In order to use Point Cloud tracking in your own applications, you will need to implement support for it using the Point Cloud API. The following section provides an overview of the system architecture as well as specific detail about the API.

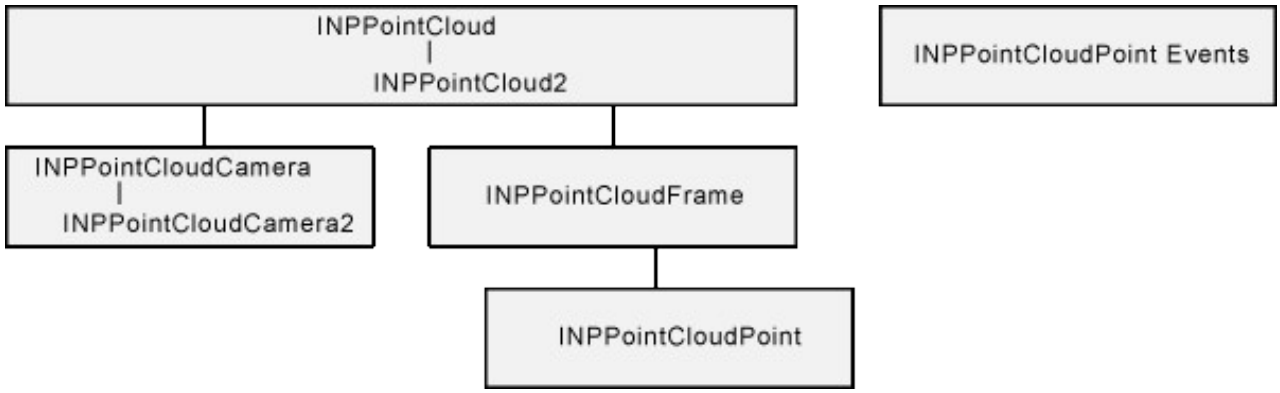
5.1 Architecture

The Point Cloud API is written as a series of COM automation interfaces and is built on top of the OptiTrack SDK. This type of interface was chosen due to the flexibility it provides. COM automation interfaces are supported by nearly every language, including VBScript, JavaScript, Visual Basic and C/C++. Support also exists for Python, Delphi and many others. Check your favorite language for more details.



5.2 Interface Layout

The figure below shows the interfaces of the Point Cloud API and their relationships.



The main interface is INPPointCloud. This interface provides the ability to enumerate the cameras, load calibration profiles, and process tracking data. When a frame of 3D data is ready, the client will obtain an INPPointCloudFrame object, markers detected in the frame are available via INPPointCloudPoint. Notifications about when a new frame of 3D data is available are received through the _INPPointCloudEvents interface.

5.3 Design Considerations

5.3.1 Getting Started

Only a small amount of code needs to be written in order to begin. The following outlines the procedure for initializing the camera:

1. Make sure the cameras have been calibrated with the result saved to a file as described in Section 4.2 ("Camera Calibration tool").
2. Create the INPPointCloud object.
3. Load an existing calibration profile using the INPPointCloud->LoadProfile() method.
4. Call the INPPointCloud->Start() method to enumerate and start the cameras.

At this point, the cameras should be initialized and collecting frame information. The main loop of the application should either handle frame callbacks using the _INPPointCloudEvents interface or poll for frames using INPPointCloud->GetFrame().

When data processing is complete, call the INPPointCloud->Stop() method.

5.3.2 Tracking Markers

The Point Cloud API provides a simple interface to the powerful functionality which collects 2D tracking data from multiple OptiTrack cameras and merges it into 3D position data. To create a single frame of 3D data the Point Cloud software will collect one frame from each camera present. When all of the frames are collected and the 3D position data has been calculated, a OnFrameAvailable() callback will be triggered to notify the client application. The client can also choose to poll for new frames using INPPointCloud->GetFrame(). The client will then obtain an INPPointCloudFrame object for the frame.

This allows it to enumerate the individual markers as INPPointCloudPoints using the Item() call. The X, Y and Z axis position of the markers can be extracted at this time for use in the client application.

Frame Object Lifetime
Frame objects in the Point Cloud system are a limited resource. Be sure to call the Free() method to release the INPPointCloudFrame interface as quickly as possible.

Coordinate System
All position data will be returned in meters. The absolute accuracy of the coordinate system relies on a good calibration and the accurate use of a ground plane as described in Section 4.2 ("Camera Calibration tool").

5.3.3 Reading Camera Positions
Once a calibration is loaded, the position and orientation will be known for every camera that is connected as long as it was present during the calibration process. The position data can be extracted by enumerating the cameras through the GetCamera() call of the INPPointCloud interface. This returns an INPPointCloudCamera object which can be used to read X, Y, Z and the rotational data.

5.3.4 Connection Points
Standard COM connection points are used for callback notifications. Connection points were chosen because they are compatible with most languages that communicate with COM automation interfaces. See the COM documentation in the MSDN help for more information. Specifically, search for IConnectionPoint and IConnectionPointContainer. There is slightly more overhead involved with setting up connection points in C/C++, but sample code is provided to assist in coding.

5.3.5 Threading Issues
The OptiTrack APIs are apartment threaded. Apartment threaded means that only one thread can be used to call into the DLL at any given time. Any other threads that are used to call into the DLL while another thread is executing in the DLL will be blocked until the first thread completes. This also affects how callbacks are implemented. Connection point callbacks must be done on the same thread that created the object. This is accomplished by creating a hidden window whenever an object that has connection points is created. The hidden window uses the message queue of the calling thread. Callbacks are done by posting a message to the hidden window and then calling the connection point callback interface. The drawback to this mechanism is that the message queue for the thread that created the object needs to be free to run if OptiTrack notifications are to be handled in the timely fashion. For instance, if the main thread of a GUI application is used to create the OptiTrack camera object, make sure that no blocking operations are run on that thread. If a blocking operation is required, use a message loop to handle messages.

This issue only affects connection point callbacks. If camera frame information is gathered by polling this INPPointCloud interface, this is not a concern.

5.4 Interfaces

5.4.1 INPPointCloud
This is the main entry point for communicating with the Point Cloud toolkit. This interface provides the ability to enumerate cameras, load calibration profiles, and process tracking data.

5.4.1.1 INPPointCloud::LoadProfile( )

The LoadProfile( ) method attempts to load a calibration profile stored in a file. It is passed the path to the profile and returns information about whether or not it succeeded.

HRESULT LoadProfile(BSTR Filename);

Parameters

- Filename
  [in] the path to the location of the calibration profile

Parameters

Value	**Meaning**
S_OK	Method succeeded
NP_POINTCLOUD_LOAD_FAILED	Method failed, profile not loaded

5.4.1.2 INPPointCloud::Start( )

The Start( ) method enumerates and starts the cameras so that tracking will begin. This method should be called after the LoadProfile( ) method.

HRESULT Start( );

Parameters

- *none*

Parameters

Value	**Meaning**
S_OK	Method succeeded
NP_POINTCLOUD_FAILED	Method failed, cameras could not be started

5.4.1.3 INPPointCloud::Stop( )

The Stop( ) method stops the cameras so that tracking will cease.

HRESULT Stop( );

Parameters

- *none*

Parameters

Value	Meaning
S_OK	Method succeeded
NP_POINTCLOUD_FAILED	Method failed, cameras could not be stopped

5.4.1.4 INPPointCloud::GetFrame( )

The GetFrame( ) method retrieves a INPPointCloudFrame object containing a processed frame of 3D tracking data.

HRESULT GetFrame(INPPointCloudFrame ** NPPointCloudFrame);

Parameters

- NPPointCloudFrame
  [out] pointer that receives an INPPointCloudFrame interface. Return value will be NULL if no frame is available.

Parameters

Value	Meaning
S_OK	Method succeeded

5.4.1.5 INPPointCloud::GetCamera( )

The GetCamera( ) method retrieves a INPPointCloudCamera object for one of the connected OptiTrack cameras.

HRESULT GetCamera([in] LONG Index, [out] INPPointCloudCamera ** NPPointCloudCamera);

Parameters

- Index
  [in] Index of the item to retrieve.

- NPPointCloudCamera
  [out] pointer that receives a INPPointCloudCamera interface. Return value will be NULL if the camera does not exist.

Parameters

Value	Meaning
S_OK	Method succeeded

5.4.1.6 INPPointCloud::get_CameraCount( )

The get_CameraCount( ) method retrieves the number of connected OptiTrack cameras.

HRESULT get_CameraCount([out, retval] LONG* pVal);

Parameters

- pVal
  [out] pointer that receives a LONG with the number of cameras connected.

Parameters

Value	Meaning
S_OK	Method succeeded

5.4.1.7 INPPointCloud2::GetCamera( )

The INPPointCloud2 GetCamera( ) method retrieves a INPPointCloudCamera2 object for one of the connected OptiTrack cameras. The INPPointCloud2 version of this method uses a retval unlike the INPPointCloud one.

HRESULT GetCamera([in] LONG Index, [out, retval] INPPointCloudCamera2 ** NPPointCloudCamera);

Parameters

- Index
  [in] Index of the item to retrieve.

- NPPointCloudCamera2
  [out, retval] pointer that receives a INPPointCloudCamera2 interface. Return value will be NULL if the camera does not exist.

Parameters

Value	Meaning
S_OK	Method succeeded

5.4.2 INPPointCloudFrame

This object contains information about the current frame of processed 3D tracking data. This interface is a standard COM enumeration interface containing a list of all objects in the frame.

5.4.2.1 INPPointCloudFrame::get_Count( )

The get_Count( ) method retrieves the number of 3D markers detected in the frame.

HRESULT get_Count([out, retval] LONG* pVal);

Parameters

- pVal
  [out] pointer that receives a LONG with the number of 3D markers in the frame.

Parameters

Value	Meaning
S_OK	Method succeeded

5.4.2.1 INPPointCloudFrame::Item( )

The Item( ) allows the enumeration of the 3D markers detected in a frame.

HRESULT Item([in] LONG a_vlIndex, [out, retval] INPPointCloudPoint ** NPPointCloudPoint);

Parameters

- Index
  [in] Index of the item to retrieve.

- NPPointCloudPoint
  [out, retval] pointer that receives a INPPointCloudPoint interface. Return value will be

NULL if the marker does not exist.

Parameters

Value	Meaning
S_OK	Method succeeded

5.4.3 INPPointCloudPoint
This object contains 3D (X,Y,Z) position information about a single detected marker in a frame.

5.4.3.1 INPPointCloudPoint::get_X( )

The get_X( ) method retrieves the X axis position of a 3D marker detected in the frame.

HRESULT get_X([out, retval] DOUBLE* pVal);

Parameters

- pVal
  [out] pointer that receives a Double with the X axis position in meters of a 3D marker.

Parameters

Value	Meaning
S_OK	Method succeeded
E_POINTER	Pointer is invalid

5.4.3.2 INPPointCloudPoint::get_Y( )

The get_Y( ) method retrieves the Y axis position of a 3D marker detected in the frame.

HRESULT get_Y([out, retval] DOUBLE* pVal);

Parameters

- pVal
  [out] pointer that receives a Double with the Y axis position in meters of a 3D marker.

Parameters

Value	Meaning
S_OK	Method succeeded

E_POINTER	Pointer is invalid

5.4.3.3 INPPointCloudPoint::get_Z( )

The get_Z( ) method retrieves the Z axis position of a 3D marker detected in the frame.

HRESULT get_Z([out, retval] DOUBLE* pVal);

Parameters

- pVal
  [out] pointer that receives a Double with the Z axis position in meters of a 3D marker.

Parameters

Value	**Meaning**
S_OK	Method succeeded
E_POINTER	Pointer is invalid

5.4.4 INPPointCloudCamera
This object contains 3D (X,Y,Z) position and orientatation (rotation) information about a connected OptiTrack camera.

5.4.4.1 INPPointCloudCamera::get_X( )

The get_X( ) method retrieves the X axis position of a camera.

HRESULT get_X([out, retval] DOUBLE* pVal);

Parameters

- pVal
  [out] pointer that receives a Double with the X axis position in meters of a camera.

Parameters

Value	**Meaning**
S_OK	Method succeeded
E_POINTER	Pointer is invalid

5.4.4.2 INPPointCloudCamera::get_Y( )

The get_Y( ) method retrieves the Y axis position of a camera

HRESULT get_Y([out, retval] DOUBLE* pVal);

Parameters

- pVal
  [out] pointer that receives a Double with the Y axis position in meters of a camera.

Parameters

Value	Meaning
S_OK	Method succeeded
E_POINTER	Pointer is invalid

5.4.4.3 INPPointCloudCamera::get_Z( )

The get_Z( ) method retrieves the Z axis position of a camera

HRESULT get_Z([out, retval] DOUBLE* pVal);

Parameters

- pVal
  [out] pointer that receives a Double with the Z axis position in meters of a camera.

Parameters

Value	Meaning
S_OK	Method succeeded
E_POINTER	Pointer is invalid

5.4.4.4 INPPointCloudCamera::GetRotationMatrix( )

The GetRotationMatrix( ) method retrieves one element from the Camera Rotation Matrix array.

HRESULT GetRotationMatrix([in] LONG Index, [out, retval] DOUBLE* pVal);

Parameters

- Index
  [in] Index of the item in the array to retrieve. There are 9 elements in the array, valid inputs range from 0 to 9.

- pVal
  [out] pointer that receives a Double with the selected item of the Camera Rotation Matrix array.

Parameters

Value	Meaning
S_OK	Method succeeded
E_POINTER	Pointer is invalid

5.4.4.5 INPPointCloudCamera2::GetOptiTrackCamera( )

The GetOptiTrackCamera( ) method retrieves a OptiTrack INPCamera object. It is only available when using INPPointCloudCamera2.

HRESULT GetOptiTrackCamera([out, retval] INPCamera ** NPCamera);

Parameters

- Index
- pVal
  [out, retval] pointer that receives a OptiTrack INPCamera interface. Return value will be NULL if the camera does not exist.

Parameters

Value	Meaning
S_OK	Method succeeded

5.5 Return Codes

The following is a listing of non-successful return codes for the Point Cloud API.

```
enum __MIDL___MIDL_itf_PointCloudCOM_0000_0001
{
     NP_POINTCLOUD_FILE_NOT_FOUND = 1,
     NP_POINTCLOUD_LOAD_FAILED = NP_POINTCLOUD_FILE_NOT_FOUND + 1,
     NP_POINTCLOUD_FAILED = NP_POINTCLOUD_LOAD_FAILED + 1,
     NP_POINTCLOUD_CAMERAS_ALREADY_STARTED = NP_POINTCLOUD_FAILED + 1,
     NP_POINTCLOUD_CAMERAS_NOT_RUNNING = NP_POINTCLOUD_CAMERAS_ALREADY_STARTED + 1,
```

```
        NP_POINTCLOUD_PROFILE_ALREADY_LOADED = NP_POINTCLOUD_CAMERAS_NOT_RUNNING + 1,
        NP_POINTCLOUD_CAMERAS_RUNNING = NP_POINTCLOUD_PROFILE_ALREADY_LOADED + 1,
        NP_POINTCLOUD_INVALID_FILE = NP_POINTCLOUD_CAMERAS_RUNNING + 1,
        NP_POINTCLOUD_INVALID_PROFILE_CAMERA_COUNT = NP_POINTCLOUD_INVALID_FILE + 1,
        NP_POINTCLOUD_UNABLE_TO_INITIALIZE_OPTITRACK_CAMERAS =
          NP_POINTCLOUD_INVALID_PROFILE_CAMERA_COUNT + 1,
        NP_POINTCLOUD_FAILED_LICENSE_CHECK =
          NP_POINTCLOUD_UNABLE_TO_INITIALIZE_OPTITRACK_CAMERAS + 1,
        NP_POINTCLOUD_NO_PROFILE_LOADED = NP_POINTCLOUD_FAILED_LICENSE_CHECK + 1
        NP_POINTCLOUD;
}
```

6. Using the Rigid Body Software

The Rigid Body toolkit is a robust, real-time 3D optical tracking solution. Markers can be attached to multiple objects in known patterns (rigid bodies) allowing them to be tracked in full 6DOF (position and orientation). Tracking data can be accessed in real time through network streaming support or the easy to use software API. The information provided below will help you get the best results from your OptiTrack cameras and Rigid Body software.

6.1 Getting Started

Before using the Rigid Body software make sure to have all of the software and hardware properly installed and licensed as described in Chapter 3. Once the installation is complete you can begin using the Rigid Body software by following these instructions :

1. Connect 3 or more cameras to the USB port of the computer and arrange them to set up a capture volume as described in Section 3.5 ("Camera Placement").

2. Calibrate the cameras using the Point Cloud calibration tool as described in Section 4.2 ("Camera Calibration Tool") and save the calibration to a file.

3. Start the Rigid Body software by clicking on the "OptiTrack Rigid Body Tool" icon on your desktop.

4. Load the calibration into the Rigid Body software to begin capturing 3D point cloud data from the cameras.

5. Rigid bodies for tracking can be defined using the 3D point cloud data, or by loading existing rigid body definitions from a file.

3D Viewport Control

The 3D viewport in the software is used to display cameras, 3D point cloud data and tracked rigid bodies. It is also used for selecting markers when defining rigid bodies. The mouse is used for marker selection and 3D viewport manipulation.

- The Left mouse button is used for marker selection.
- The Right mouse button rotates the 3D view when the button is held down and mouse is moved.
- The Scroll Wheel on the mouse zooms the 3D view in and out.
- The Middle button on the mouse pans/translates the 3D view when the button is held down and mouse is moved.

6.2 Working with Rigid Bodies

The OptiTrack Rigid Body software creates a cloud of 3D points representing reflective markers which are visible to cameras within the capture volume. Rigid Bodies are clusters of reflective markers in a unique configuration which allow them to be identified and tracked in the cloud of 3D points. It is possible to track multiple rigid bodies at a time in full 6DOF (*position and orientation*).

The following section introduces best practices for defining rigid bodies and explains how to track them using the Rigid Body software.

6.2.1 Making Rigid Bodies

To get the best tracking performance there are several factors listed below to consider when making the physical rigid bodies. The configuration(s) will also be influenced by the tracking volume and camera placement, the size and shape of the physical object which the markers are mounted to, and the other objects which may be present within the tracking volume.

Marker Types

Spherical reflective markers will usually yield the most stable and accurate 3D tracking data. Hemispheric and flat markers are less desirable because their shape as it appears to the camera may change when it rotates, this can introduce errors when calculating their center of mass and position.

When using small rigid bodies, try to use small markers to reduce the chances of one marker occluding (*blocking from view of a camera*) other markers.

When tracking at larger distances from the camera, using larger markers can help improve the resolution of the tracking. Marker sizes typically range from 10 to 25 millimeters and larger.

NaturalPoint offers a range of high quality pre-made reflective markers for sale, for more details see Section 9 ("Cameras and Accessories").

Marker Quantity

Three markers is the minimum required to define a rigid body. A larger number of

markers can be used to increase the precision and reduce likelihood of the rigid body flipping. Using more than three markers also provides redundancy, then tracking can occur even when some of the assigned markers are not visible (*the minimum visible for tracking is three*).

Rigid Body Sizing
The physical arrangement of the markers should not be too small and should not have markers too close together. Markers closer than 6 millimeters can result in the following issues :

- Markers are more likely to overlap or occlude one another when seen from a camera, resulting in incorrect 3D position data or markers failing to be tracked.
- If the point cloud tracking residual value is too high, or the calibration is poor, then rays for adjacent markers could be grouped together incorrectly resulting in misidentified marker locations and/or falsely reported markers.

Marker Arrangement
For 3-marker rigid bodies, using asymmetrical marker arrangements improves the tracking reliability. Symmetrical triangles make it harder for the software to identify the correct orientation of a rigid body and increases the likelihood of arriving at an incorrect solution . This can often manifest as the rigid body flipping on its axis between frames.

When multiple rigid bodies will be tracked at the same time, avoid making rigid bodies which are too similar to each other. Making them individually unique with different marker arrangements and sizes will reduce the likelihood of misidentification and swapping.

6.2.2 Defining and Deleting Rigid Bodies

Before you can begin tracking rigid bodies, you will need to create them in the software using point cloud data.

The first step is to load previously saved point cloud data or load a calibration and begin capturing new point cloud data. Once there is point cloud data to work from, the easiest way to redefine a rigid body is to select the markers in the viewport and press the "Add Rigid Body" button in the panel to the right. This will create a new rigid body in the shape of the selected markers. Selecting markers in the viewport will automatically pause the real-time capture or playback.

Multiple markers can be selected by holding the **shift** key down and drawing a rectangle over them using the left mouse button. Individual markers can be selected and de-selected by holding the **ctrl** key down while clicking on markers with the left mouse button. The color of the markers will change once they are selected.

A rigid body which uses all of the visible markers in the 3D viewport can be created by clicking the "Create From Visible Markers" button.

To preview the new rigid body, resume playback of the recorded or live data and the rigid body will now begin tracking. As rigid bodies are created, they will show up in a list in the panel to the right.

Rigid bodies can be removed by clicking on the name of a rigid body and pressing the "Delete Rigid Body" button.

6.2.3 Editing Rigid Bodies

Once a rigid body has been created, it can be modified at a later time by selecting its name from the list of rigid bodies.

- **Properties** : The name, ID, display color and tracking flexibility can be changed for a rigid body. The tracking flexibility controls how much the shape of the rigid body can distort before it will no longer track. Increasing the tolerance will allow the rigid body to track better if the physical shape is distorted, but can also result in the wrong markers being tracked for the rigid body or solving for the orientation incorrectly. The valid range of flexibility values are from 0 to 1.0 with a value of 1.0 being more flexible.

- **Movement Constraints** : Constraints on how fast a rigid body can translate (move in X,Y,Z) and rotate (heading, attitude, bank) can be set. This allows the limitations of how fast the physical rigid body can translate and rotate to be imposed on the rigid body tracking in the software. Tuning these constraints can help improve the robustness of the tracking.

  Rotation constraint values are in angular degrees per frame, with a range from 0 to 180 degrees. A value of 0 disables the rotation constraint. Translation constraint values are in meters per frame with a value of 0 disabling the constraint.

- **Marker List** : This list contains all of the markers used in the rigid body and their positions. The point of origin for a rigid body can be changed by selecting a marker in the list and pressing the "Set Origin" button. Markers can also be removed from a rigid body by selecting a marker in the list and pressing the "Delete" button.

- **3D Model** : Custom 3D models can be loaded and assigned to a rigid body for more intuitive feedback. A model assigned to a rigid body will translate and rotate in the 3D viewport based on the motion of the associated rigid body.

  3D Model Translation is in meters, 3D Model Rotation ranges from -180 to +180 and is adjusted by the slider.

6.2.4 Loading and Saving Rigid Bodies

Rigid Body definitions created in the software can be saved to a file for later use or manual editing, and easily loaded back into the software.

To save all of the rigid bodies currently defined in the software, follow the steps below.

1. From the File menu choose "Save Rigid Body Definitions"
2. Enter a filename and press the "Save" button

To load previously defined rigid bodies into the software, follow the steps below. Note that loading a set of rigid body definitions will replace any rigid bodies currently defined in the software. Once loaded, the rigid bodies will show up in a list in the panel to the right.

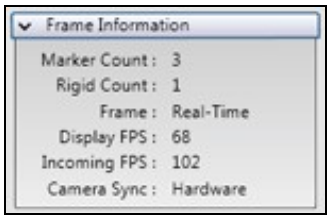
1. From the File menu choose "Load Rigid Body Definitions"
2. Select the file with rigid body definitions and press the "Open" button

6.3 Tracking and Recording

The OptiTrack Rigid Body software provides real-time tracking with the ability to record 3D point cloud data and play it back offline. The following section covers data preview, capture and export to other applications.

6.3.1 Tracking Status

Status about the cameras, 3D point cloud reconstruction and playback can be found in Frame Information area. The feedback provided here can be helpful in assessing the health of the system and troubleshooting.



- **Marker Count** : Indicates the number of markers contained in the current frame of the 3D point cloud.
- **Rigid Count** : Indicates the number of rigid bodies found within the current frame of the 3D point cloud. Rigid bodies defined in the software which are not found in the current frame will not be included in the total.
- **Frame** : During playback of recorded data this will display the current frame number. When operating in real-time mode with the cameras, the value "Real-Time" will be displayed instead.
- **Display FPS** : Indicates the rate in frames-per-second at which the 3D viewport is being updated.
- **Incoming FPS** : Indicates the rate in frames-per-second which 3D point cloud data is being processed. When using V100 cameras this number should close to "100". Large quantities of markers and rigid bodies can load the system down and may reduce this number, indicating that some frames are being skipped or dropped.
- **Camera Sync** : Indicates the synchronization status of the cameras. When the camera-sync cables are connected and the cameras are operating in hardware synchronization mode, the value "Hardware" will be displayed. Using the cameras in hardware synchronization mode will produce the best quality 3D point cloud data. If the camera-sync cables are not connected between all cameras then the software will attempt to synchronize the cameras using the time of delivery for each frame. When software synchronization is being performed, the value "Synchronized" will be displayed.

6.3.2 Real-time Tracking

Processing and displaying live 3D point cloud data from the cameras as it is captured is referred to as real-time operation. As markers move within the field of view of the

cameras, a 3D point cloud of their current positions is created. The 3D point cloud is searched to determine the location of currently defined rigid bodies. This data can be recorded for later use, or streamed to other applications as it comes in from the cameras.

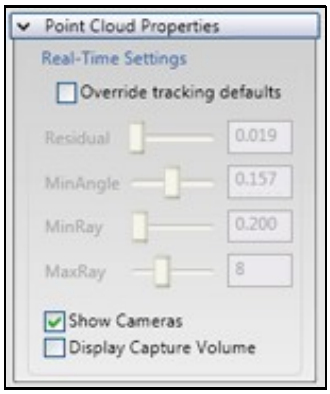
Loading Camera Calibrations and Starting the Cameras
Before the software can begin tracking in real-time the cameras need to be calibrated using the Point Cloud calibration tool as described in Section 4.2 ("Camera Calibration Tool"). Once the calibration has been completed it should be saved to a file. This file can then be loaded into the Rigid Body software by choosing "Load Calibration Profile" from the Tracking menu. As soon as the calibration profile is loaded the software will begin creating the 3D point cloud and displaying it in the viewport.

To stop collecting tracking data from the cameras, choose "Shutdown Cameras" from the Tracking menu. After as shutdown has been performed, a calibration file will have to be loaded again in order to resume tracking.

3D Point Cloud
The point cloud engine collects 2D marker data from the individual cameras and reconstructs it into a 3D point cloud containing the identified markers. The rigid body solver then searches this 3D point cloud to find any of the currently defined rigid bodies.

Cameras which observe the same physical markers from different angles can use known information about their positions to triangulate where the marker is in 3D space. The imaginary lines drawn from the focal point of a camera to the projected location of each 2D marker in its field of view are called Rays. 3D points are formed when multiple rays from different cameras cross within a specified distance of each other. A ray which does not cross with other rays, or which crosses at too great of a distance from other rays will not be used for creating 3D markers.

The default settings for point cloud tracking are usually sufficient, however they can be modified to tune the performance for specific camera and rigid body configurations. To change these settings, open the Point Cloud Properties tab and check the "Override Tracking defaults" checkbox.



- **Residual** : Rays from the camera to the marker must cross closer than this value (in meters) to form 3D points. Making this value too large can result in an increased numbers of falsely detected 3D markers. Making the value too small can prevent 3D markers from being formed, especially if the cameras are poorly calibrated.

- **Min Angle** : Rays from the camera to the marker must cross at an angle (in degrees) larger than this to form 3D points. If cameras observing the same marker are positioned very closely together, the this value may need to be reduced.

- **Min Ray** : Rays from the camera to the marker must cross farther than this distance (in meters) from the camera to form 3D points.

- **Max Ray** : Rays from the camera to the marker must cross closer than this distance (in meters) from the camera to form 3D points.

- **Show Cameras** : Controls whether 3D representations of the cameras are shown in the 3D viewport.

- **Display Capture Volume** : This provides a way to visualize the physical area of the capture volume. It is generated by estimating the 3D region where the field of view of multiple cameras overlap. It can be used to determine which areas have better tracking coverage.

6.3.3 Recording and Playback

The real-time 3D point cloud data from the cameras can be recorded and played back offline. When using recorded data, rigid bodies are tracked in the same way as when live data from the cameras is used. Recorded data provides the ability to play the data backward as well as review a specific set of motion in close detail as many times as needed. Since the recorded data can be streamed to other applications, it can also be used in testing and validation for downstream tools which consume the rigid body tracking data.

Playback Button Controls

1. **Rewind to the First Frame**
2. **Play Backward**
3. **Stop**
4. **Pause**
5. **Play Forward**
6. **Advance to the Last Frame**
7. **Record**

How to Record Tracking Data
Before 3D point cloud data can be recorded, make sure that a calibration file has been loaded and the cameras are tracking as described in Section 6.3.2 ("Real-time Tracking").

To begin recording, press the record button. When the desired amount of data has been captured, recording can be stopped by pressing either the Record button again, or the Stop button. To save the data that was recorded, choose "Save Capture Data" from the File menu, enter a file name and press the "Save" button.

Playing Back Recorded Data
If recorded 3D point cloud data is already loaded, then playback can be started by pressing the Play button. To play the data in reverse, press the Play Backward button. If data has not yet been loaded then choose "Load Capture Data" from the File menu, select a file and press the "Open" button.

While data is being played back the software will display it in the 3D viewport and the timeline scrubber will advance. The current frame number will be displayed in the "Frame Information" tab (if the tab is expanded). Playback can be stopped using the Pause or Stop buttons.

Switching Between Real-time and Recorded Data Playback
The Rigid Body software makes it easy to switch between capturing real-time data and playing back recorded data. The value "Real-time" will be displayed when the software is in real-time mode, otherwise it will display "Frame" plus the current frame number.

If recorded data is currently loaded in the software, pressing either of the Play buttons will begin playing it back. Using the Pause button will momentarily halt playback, but will not switch to real-time mode. Pressing the Stop button switches the software into real-time mode until any of the other playback controls are pressed again.

Editing Recorded Data
The Rigid Body software provides a set of basic tools for editing recorded data. These tools are located under the "Recorded Data" section of the Edit menu.

- **Delete All Frames** : Removes all of the recorded frame data.
- **Delete Before Current Frame** : Removes all of the recorded frame data before the currently displayed frame.
- **Delete After Current Frame** : Removes all of the recorded frame data after the currently displayed frame.

Recording and Playback Options
A number of options for recording and playback are available, they can be found under the "Recording and Playback" tab.

- **Playback Rate** : The speed at which the recorded 3D point cloud data is played back can be adjusted. Setting the value to smaller numbers will make the data play back slower, larger numbers will result in accelerated playback. A value of 50% is 1/2 the real speed, and 200% is double the real speed.

- **Loop Playback** : When the last frame in a set of recorded data is reached, playback will continue at the first frame if this setting is checked.

- **Marker Locations** : Controls whether 3D point cloud marker locations are displayed in the 3D viewport.

- **Rigid Body Locations** : Controls whether identified rigid bodies are displayed in the 3D viewport.

- **Rigid Body Markers** : Controls whether the marker positions which a rigid body is composed of are displayed in the 3D viewport. Showing these can be helpful in determining how much the marker locations of a physical rigid body differ from the expected marker positions.

- **Rigid Body 3D Models** : Controls whether 3D models assigned to rigid bodies are displayed in the 3D viewport.

6.3.4 Exporting and Streaming Tracking Data
The 3D point cloud and rigid body tracking data can be exported using files or streamed in real-time for use in other applications. The details below explain how to use these features.

Exporting to CSV format

Recorded tracking data with rigid body locations can be exported in the CSV format for use in other applications. The exported file will contain comments at the top which describe the data formatting.

Before exporting data to a CSV file, make sure that recorded data is currently loaded and rigid bodies are defined. Then choose "Export" from the File menu, enter a file name and press the "Save" button.

Streaming tracking data

Real-time and recorded data can be streamed over the network for use in other applications. Several different methods of streaming are supported including industry standards VRPN and Trackd. All of the streaming methods are built on a network transport which allows the data to be made available on the local computer as well as remote network computers.

Before trying to stream, make sure that either recorded data is currently loaded or the cameras are tracking in real-time. It is also necessary to have rigid bodies defined.

Since data is streamed over TCP/IP on the network, it may be necessary to review your firewall settings. If the firewall is blocking traffic it may prevent the Rigid Body software from properly sending the data or the client applications from reading it.

The settings which control streaming can be found under the "Streaming" tab.

- **NaturalPoint Engine (NatNet)** : The NaturalPoint streaming engine is a custom streaming implementing which can be used to access real-time and recorded tracking data over the network. NaturalPoint provides sample source code for writing a client which receives the streaming data. The standard sample client can be used to verify that streaming is functioning properly.

  Streaming is enabled by checking the "Broadcast Frame Data" checkbox.

  Streams : rigid bodies and markers

  Network Details : port 1510, multigast group 1001, UDP multicast

- **VRPN Engine** : VRPN is an open source set of classes and a protocol which can be used to access real-time and recorded tracking data over the network. It features low latency and has built-in auto-renegotiation if a connection is temporarily dropped. NaturalPoint provides sample source code for a listener that receives tracking data. Additional sample code is available from the VRPN website .

  In VRPN objects are identified and accessed by a name. In the Rigid Body software's implementation of VRPN the name used is the one assigned to a rigid body under the "Rigid Body Definition" tab.

  Streaming is enabled by checking the "Broadcast Frame Data" checkbox.

  If needed, the VRPN port can be changed from the default by editing the port number setting.

  Streams : rigid bodies

  Network Details : selectable port, default port 3883, TCP + UDP

- **Trackd Engine** : Trackd is a standard created by VRCO/Mechdyne which can be used to access real-time and recorded tracking data over the network. NaturalPoint provides sample source code for a listener that receives tracking data. Additional information is available from the VRCO website.

  The Trackd server must be configured before data can be streamed, use the following steps to get it set up.

  1. A Trackd module (dll) and sample configuration file are provided by NaturalPoint.

  2. Configure Trackd by calling it the "naturalpointtracker".

  3. Define the number of rigid bodies desired to track.

  4. The streaming host defaults to localhost, but can be changed to a different network address.

  5. Once the configuration is complete, run the Trackd server.

  Streaming is enabled by checking the "Broadcast Frame Data" checkbox.

  Streams : rigid bodies

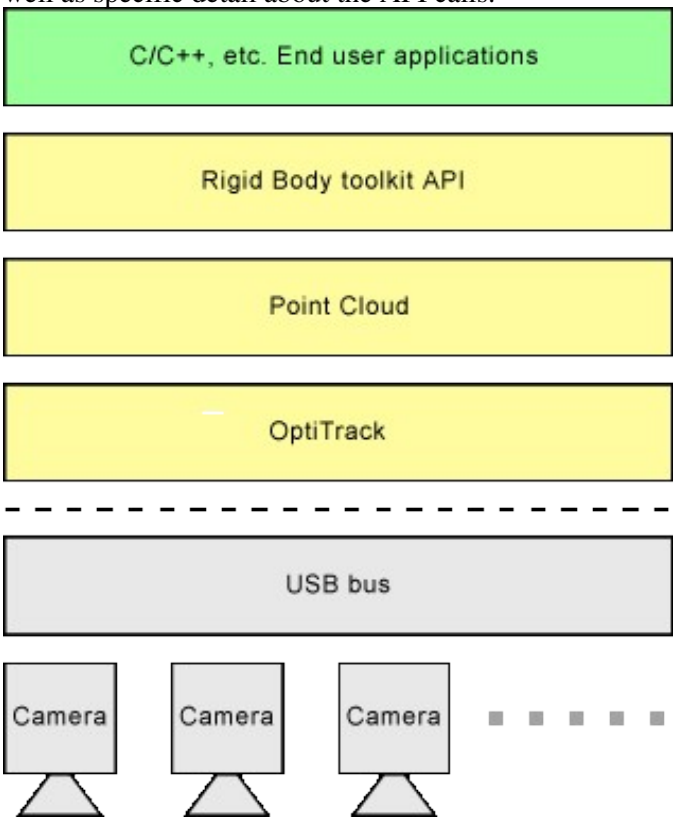
  Network Details : port 4994, TCP + UDP

6.4 Application Preferences

Several aspects of the Rigid Body software can be customized to better suit your needs. The settings which control customization are found in the Preferences window, which can be launched by choosing "Preferences" from the Edit menu. Changes to these settings will be preserved when the software is closed, and restored when it is started again.

7. Using the Rigid Body API

In order to use Rigid Body tracking in your own applications, you will need to implement support for it using the Rigid Body API. The Rigid Body API is written as a set of C/C++ function calls and a loadable DLL. The following section provides an overview of the system architecture as well as specific detail about the API calls.



7.1 Getting Started

Only a small amount of code needs to be written in order to begin tracking rigid bodies. The following outlines the initialization procedure :

1. Make sure the cameras have been calibrated with the result saved to a file as described in Section 4.2 ("Camera Calibration tool").

2. Initialize the rigid body tracking using RB_InitializeRigidBody().

3. Load an existing calibration profile using RB_LoadProfile().

4. Load existing rigid body definitions using RB_LoadDefinition().

5. Start the cameras using RB_StartCameras().

At this point, the cameras should be initialized and collecting frame information. The main loop of the application should poll for frames using RB_GetNextFrame() or RB_GetLatestFrame().

When data processing is complete, call the RB_StopCameras() and then RB_ShutdownRigidBody.

7.2 Rigid Body API Calls

7.2.1 Rigid Body Startup and Shutdown

7.2.1.1 RB_InitializeRigidBody( )

The RB_InitializeRigidBody( ) function attempts to initialize the rigid body tracking API. It should be called before attempting to use other components of the API. It returns information about whether or not it succeeded.

NPRESULT RB_InitializeRigidBody()

Parameters

· none

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded

7.2.1.2 RB_ShutdownRigidBody( )

The RB_ShutdownRigidBody( ) function attempts to shutdown the rigid body tracking API. It should be called after RB_StopCameras() and before the application using the rigid body API closes. It returns information about whether or not it succeeded.

NPRESULT RB_ShutdownRigidBody()

Parameters

· none

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded

7.2.2 Rigid Body Interface

7.2.2.1 RB_LoadProfile()

The RB_LoadProfile() function attempts to load a calibration profile stored in a file. It is passed the path to the profile and returns information about whether or not it succeeded.

NPRESULT RB_LoadProfile(const char *filename)

Parameters

· filename
[in] the path to the location of the calibration profile

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
InvalidFile	The file or path specified is not valid
InvalidLicense	A valid Rigid Body license was not found
InvalidProfileCameraCount	The specified profile does not match the current camera configuration

7.2.2.2 RB_StartCameras()

The RB_StartCameras() function starts the cameras so that tracking will begin. This function should be called after RB_LoadProfile(). It returns information about whether or not it succeeded.

NPRESULT RB_StartCameras()

Parameters

· none

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
PointCloudNotInitialized	The Point Cloud engine has not been properly initialized
CamerasAlreadyStarted	The cameras have already been started
NoProfileLoaded	A calibration profile for the cameras has not been loaded
UnableToInitializeOptiTrackCameras	A problem was encountered when initializing the cameras

7.2.2.3 RB_StopCameras()

The RB_StopCameras() function stops the cameras so that tracking will cease. It returns information about whether or not it succeeded.

NPRESULT RB_StopCameras()

Parameters

· none

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
PointCloudNotInitialized	The Point Cloud engine has not been properly initialized
CamerasNotRunning	The cameras are already stopped

7.2.2.4 RB_GetNextFrame()

The RB_GetNextFrame() function loads the next available frame of tracking data. Information about the frame is accessed using the Rigid Body Control set of API calls. It returns information about whether or not it succeeded.

NPRESULT RB_GetNextFrame()

Parameters

· none

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
InvalidLicense	A valid Rigid Body license was not found
NoFrameAvailable	No tracking data was found

7.2.2.5 RB_GetLatestFrame()

The RB_GetLatestFrame() function loads the most recent frame of tracking data. Information about the frame is accessed using the Rigid Body Control set of API calls. It returns information about whether or not it succeeded.

NPRESULT RB_GetLatestFrame()

Parameters

· none

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
InvalidLicense	A valid Rigid Body license was not found
NoFrameAvailable	No tracking data was found

7.2.2.6 RB_LoadDefinition()

The RB_LoadDefinition() function loads a set of rigid body definitions from a file for tracking. It returns information about whether or not it succeeded.

NPRESULT RB_LoadDefinition(const char *filename)

Parameters

· filename

[in] the path to the location of the rigid body definition file

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
Failed	The definitions failed to load

7.2.2.7 RB_SaveDefinition()

The RB_SaveDefinition() function saves the currently loaded rigid body definitions to a file for later use. It returns information about whether or not it succeeded.

NPRESULT RB_SaveDefinition(const char *filename)

Parameters

· filename
[in] the path to the location of the rigid body definition file

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded
Failed	Saving the definitions failed

7.2.3 Rigid Body Streaming

7.2.3.1 RB_StreamTrackd()

The RB_StreamTrackd() function starts and stops streaming tracking data over the Trackd interface. It returns information about whether or not it succeeded.

NPRESULT RB_StreamTrackd(bool enabled)

Parameters

· enabled
[in] starts and stops the streaming, true starts and false stops

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded

7.2.3.2 RB_StreamVRPN()

The RB_StreamVRPN() function starts and stops streaming tracking data over the VRPN interface. It returns information about whether or not it succeeded.

NPRESULT RB_StreamVRPN(bool enabled, int port)

Parameters

- enabled
  [in] starts and stops the streaming, true starts and false stops
- port
  [in] selects the network port to broadcast over

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded

7.2.3.3 RB_StreamNP()

The RB_StreamNP() function starts and stops streaming tracking data over the NaturalPoint streaming interface. It returns information about whether or not it succeeded.

NPRESULT RB_StreamNP(bool enabled)

Parameters

- enabled
  [in] starts and stops the streaming, true starts and false stops

Parameters

Value	Meaning
NPRESULT_SUCCESS	Method succeeded

7.2.4 Rigid Body Frame

7.2.4.1 RB_FrameMarkerCount()

The RB_FrameMarkerCount() function returns the number of 3D point cloud markers found in the current frame.

int RB_FrameMarkerCount()

Parameters

- (returns)
  [out] the number of 3D point cloud markers found in the current frame

7.2.4.2 RB_FrameMarkerX()

The RB_FrameMarkerX() function returns the X axis position in meters of the selected 3D point cloud marker.

float RB_FrameMarkerX(int index)

Parameters

- (returns)
  [out] the X position in meters of the selected 3D point cloud marker
- index
  [in] the index number of the desired marker. the index is zero based

Parameters

Value	Meaning
0	No frame data was found

7.2.4.3 RB_FrameMarkerY()

The RB_FrameMarkerY() function returns the Y axis position in meters of the selected 3D point cloud marker.

float RB_FrameMarkerY(int index)

Parameters

- (returns)
  [out] the Y position in meters of the selected 3D point cloud marker
- index
  [in] returns the index number of the desired marker. the index is zero based

Parameters

Value	Meaning
0	No frame data was found

7.2.4.4 RB_FrameMarkerZ()

The RB_FrameMarkerZ() function returns the Z axis position in meters of the selected 3D point cloud marker.

float RB_FrameMarkerZ(int index)

Parameters

- (returns)
  [out] the Z position in meters of the selected 3D point cloud marker
- index
  [in] returns the index number of the desired marker. the index is zero based

Parameters

Value	Meaning
0	No frame data was found

7.2.5 Rigid Body Control

7.2.5.1 RB_IsRigidBodyTracked()

The RB_IsRigidBodyTracked() function returns information about whether the selected rigid body is found in the current frame.

bool RB_IsRigidBodyTracked(int index)

Parameters

- (returns)
  [out] true if the selected rigid body is found in the current frame
- index
  [in] the index number of the desired rigid body. the index is zero based

7.2.5.2 RB_GetRigidBodyLocation()

The RB_GetRigidBodyLocation() function returns the position and orientation of the selected rigid body. RB_IsRigidBodyTracked() should be checked first to determine whether the rigid body was tracked in the current frame, otherwise the data may be stale.

void RB_GetRigidBodyLocation(int RigidIndex,
float *x, float *y, float *z,
float *qx, float *qy, float *qz, float *qw,
float *heading, float *attitude, float *bank)

Parameters

- RigidIndex
  [in] the index number of the desired rigid body. the index is zero based
- x
  [out] receives the X axis position in meters of the selected rigid body
- y
  [out] receives the Y axis position in meters of the selected rigid body
- z
  [out] receives the Z axis position in meters of the selected rigid body
- qx
  [out] receives the quaternion x value of the selected rigid body
- qy
  [out] receives the quaternion y value of the selected rigid body
- qz
  [out] receives the quaternion z value of the selected rigid body
- qw
  [out] receives the quaternion w value of the selected rigid body
- heading
  [out] receives the heading orientation value in degrees of the selected rigid body
- attitude
  [out] receives the attitude orientation value in degrees of the selected rigid body
- bank
  [out] receives the bank orientation value in degrees of the selected rigid body

7.2.5.3 RB_ClearRigidBodyList()

The RB_ClearRigidBodyList() function removes all of the currently loaded rigid body definitions.

void RB_ClearRigidBodyList()

Parameters

· none

7.2.5.4 RB_GetRigidBodyCount()

The RB_GetRigidBodyCount() function returns the number of currently loaded rigid body definitions.

int RB_GetRigidBodyCount()

Parameters

· (returns)
[out] the number of currently loaded rigid body definitions.

7.2.5.5 RB_DeleteRigidBodyMarker()

The RB_DeleteRigidBodyMarker() function deletes a marker from a rigid body definition.

void RB_DeleteRigidBodyMarker(int RigidIndex, int MarkerIndex)

Parameters

· RigidIndex
[in] the index of the desired rigid body to remove the marker from. the index is zero based
· MarkerIndex
[in] the index of the desired marker to remove. the index is zero based

7.2.5.6 RB_AddRigidBodyMarker()

The RB_AddRigidBodyMarker() function adds a marker to an existing rigid body definition.

void RB_AddRigidBodyMarker(int RigidIndex, float x, float y, float z)

Parameters

· RigidIndex
[in] the index of the desired rigid body. the index is zero based
· x
[in] the X axis position of the marker in meters relative to the origin

· y
[in] the Y axis position of the marker in meters relative to the origin
· z
[in] the Z axis position of the marker in meters relative to the origin

7.2.5.7 RB_SetRigidBodyOrigin()

The RB_SetRigidBodyOrigin() function changes the origin for an existing rigid body definition.

void RB_SetRigidBodyOrigin(int RigidIndex, float x, float y, float z)

Parameters
· RigidIndex
[in] the index of the desired rigid body. the index is zero based
· x
[in] the new X axis position of the origin in meters
· y
[in] the new Y axis position of the origin in meters
· z
[in] the new Z axis position of the origin in meters

7.2.5.8 RB_GetRigidBodyID()

The RB_GetRigidBodyID() function returns the ID for the selected rigid body.

int RB_GetRigidBodyID(int index)

Parameters
· (returns)
[out] the ID of the selected rigid body
· index
[in] the index of the desired rigid body. the index is zero based

7.2.5.9 RB_SetRigidBodyID()

The RB_SetRigidBodyID() function changes the ID for the selected rigid body.

void RB_SetRigidBodyID(int index, int ID)

Parameters

· index
[in] the index of the desired rigid body. the index is zero based
· ID
[in] the new ID value for the selected rigid body

7.2.5.10 RB_CreateRigidBody()

The RB_CreateRigidBody() function creates a new rigid body definition.

void RB_CreateRigidBody(const char *name)

Parameters
· name
[in] the name to be assigned to the newly created rigid body

7.2.5.11 RB_DeleteRigidBody()

The RB_DeleteRigidBody() function deletes an existing rigid body definition.

void RB_DeleteRigidBody(int index)

Parameters
· index
[in] the index of the rigid body to be deleted. the index is zero based

7.2.5.12 RB_GetRigidBodyFlexibility()

The RB_GetRigidBodyFlexibility() function returns the flexibility setting for the selected rigid body.

float RB_GetRigidBodyFlexibility(int index)

Parameters
· (returns)
[out] the flexibility setting for the selected rigid body
· index
[in] the index of the desired rigid body. the index is zero based

7.2.5.13 RB_SetRigidBodyFlexibility()

The RB_SetRigidBodyFlexibility() function changes the flexibility setting for the selected rigid body.

void RB_SetRigidBodyFlexibility(int index, float value)

Parameters

- index
  [in] the index of the desired rigid body. the index is zero based
- value
  [in] the new flexibility setting for the selected rigid body. the input values range from 0 (least flexible) to 1.0 (most flexible)

7.2.5.14 RB_GetRigidBodyRotationConst()

The RB_GetRigidBodyRotationConst() function returns the rotational constraint setting for the selected rigid body.

float RB_GetRigidBodyRotationConst(int index)

Parameters

- (returns)
  [out] the rotational constraint setting for the selected rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

7.2.5.15 RB_SetRigidBodyRotationConst()

The RB_SetRigidBodyRotationConst() function changes the rotational constraint setting for the selected rigid body.

void RB_SetRigidBodyRotationConst(int index, float value)

Parameters

- index
  [in] the index of the desired rigid body. the index is zero based
- value
  [in] the new rotational constraint setting for the selected rigid body. the value range is in degrees per frame and ranges from 0 to 180 degrees, a value of 0 disables the constraint

7.2.5.16 RB_GetRigidBodyTranslationConst()

The RB_GetRigidBodyTranslationConst() function returns the translation constraint setting for the selected rigid body.

float RB_GetRigidBodyTranslationConst(int index)

Parameters

- (returns)
  [out] the translation constraint setting for the selected rigid body.
- index
  [in] the index of the desired rigid body. the index is zero based

7.2.5.17 RB_SetRigidBodyTranslationConst()

The RB_SetRigidBodyTranslationConst() function changes the translation constraint setting for the selected rigid body.

void RB_SetRigidBodyTranslationConst(int index, float value)

Parameters

- index
  [in] the index of the desired rigid body. the index is zero based
- value
  [in] the new translation constraint setting for the selected rigid body. the values are in meters per frame, a value of 0 disables the constraint

7.2.5.18 RB_SetRigidBodyEnabled()

The RB_SetRigidBodyEnabled() function controls whether the selected rigid body is enabled for tracking.

void RB_SetRigidBodyEnabled(int index, bool enabled)

Parameters

- index
  [in] the index of the desired rigid body. the index is zero based
- enabled
  [in] a value of true enables the rigid body, a value of false disables it

7.2.5.19 RB_GetRigidBodyEnabled()

The RB_GetRigidBodyEnabled() function returns information about whether the

selected rigid body is enabled for tracking.

bool RB_GetRigidBodyEnabled(int index)

Parameters

- (returns)
  [out] a value of true means the rigid body is enabled, a value of false means it is disabled
- index
  [in] the index of the desired rigid body. the index is zero based

7.2.5.20 RB_SetRigidBodyColor()

The RB_SetRigidBodyColor() function changes the color used for displaying the selected rigid body in the 3D viewport.

void RB_SetRigidBodyColor(int index, int r, int g, int b)

Parameters

- index
  [in] the index of the desired rigid body. the index is zero based
- r
  [in] value for the red color component. the range is from 0 to 255
- g
  [in] value for the green color component. the range is from 0 to 255
- b
  [in] value for the blue color component. the range is from 0 to 255

7.2.5.21 RB_GetRigidBodyColorR()

The RB_GetRigidBodyColorR() function returns the red component of the color used for displaying the selected rigid body in the 3D viewport.

int RB_GetRigidBodyColorR(int index)

Parameters

- (returns)
  [out] the red color component of the rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

7.2.5.22 RB_GetRigidBodyColorG()

The RB_GetRigidBodyColorG() function returns the green component of the color used for displaying the selected rigid body in the 3D viewport.

int RB_GetRigidBodyColorG(int index)

Parameters

- (returns)
  [out] the green color component of the rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

7.2.5.23 RB_GetRigidBodyColorB()

The RB_GetRigidBodyColorB() function returns the blue component of the color used for displaying the selected rigid body in the 3D viewport.

int RB_GetRigidBodyColorB(int index)

Parameters

- (returns)
  [out] the blue color component of the rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

7.2.5.24 RB_GetRigidBodyMarkerCount()

The RB_GetRigidBodyMarkerCount() function returns the number of markers used in the selected rigid body definition.

int RB_GetRigidBodyMarkerCount(int index)

Parameters

- (returns)
  [out] the number of markers used in the selected rigid body
- index
  [in] the index of the desired rigid body. the index is zero based

7.2.5.25 RB_GetRigidBodyMarker()

The RB_GetRigidBodyMarker() function returns the position of the selected marker in the selected rigid body definition. Marker positions are in meters and

relative to the pivot point or center of mass of the rigid body.

void RB_GetRigidBodyMarker(int RigidIndex, int MarkerIndex, float *x, float *y, float *z)

Parameters

- RigidIndex
  [in] the index of the desired rigid body. the index is zero based
- MarkerIndex
  [in] the index of the desired marker. the index is zero based
- x
  [out] the X axis position of the marker in meters
- y
  [out] the Y axis position of the marker in meters
- z
  [out] the Z axis position of the marker in meters

7.2.5.26 RB_GetSyncQuality()

The RB_GetSyncQuality() function returns the current camera synchronization status.

int RB_GetSyncQuality()

Parameters

- (returns)
  [out] the current camera synchronization status. 0=No Sync, 1=Hardware Sync, 2=Good Software Sync, 3=Poor Sync

7.2.6 Point Cloud Interface

7.2.6.1 RB_SetPointCloudTrackingParams()

The RB_SetPointCloudTrackingParams() function changes the Point Cloud tracking settings.

void RB_SetPointCloudTrackingParams(int iMinRays, float fMaxResidual, float fMinAngle, float fMinRayLength, float fMaxRayLength)

Parameters

- iMinRays
  [in] the minimum number of rays required to form a 3D point cloud marker

· fMaxResidual
[in] rays from the camera to the marker must cross closer than this value (in meters) to form 3D points
· fMinAngle
[in] rays from the camera to the marker must cross at an angle (in radians) larger than this to form 3D points
· fMinRayLength
[in] rays from the camera to the marker must cross farther than this distance (in meters) from the camera to form 3D points
· fMaxRayLength
[in] rays from the camera to the marker must cross closer than this distance (in meters) from the camera to form 3D points

7.2.6.2 RB_GetPointCloudTrackingParams()

The RB_GetPointCloudTrackingParams() function returns the current Point Cloud tracking settings.

void RB_GetPointCloudTrackingParams(int* iMinRays, float* fMaxResidual, float* fMinAngle, float* fMinRayLength, float* fMaxRayLength)

Parameters

· iMinRays
[out] the minimum number of rays required to form a 3D point cloud marker
· fMaxResidual
[out] rays from the camera to the marker must cross closer than this value (in meters) to form 3D points
· fMinAngle
[out] rays from the camera to the marker must cross at an angle (in degrees) larger than this to form 3D points
· fMinRayLength
[out] rays from the camera to the marker must cross farther than this distance (in meters) from the camera to form 3D points
· fMaxRayLength
[out] rays from the camera to the marker must cross closer than this distance (in meters) from the camera to form 3D points

7.2.6.3 RB_CameraCount()

The RB_CameraCount() function returns the number of connected cameras.

int RB_CameraCount()

Parameters

· (returns)
[out] the number of connected cameras

7.2.6.4 RB_CameraXLocation()

The RB_CameraXLocation() function returns the X position of the selected camera in relation to the coordinate system origin.

float RB_CameraXLocation(int index)

Parameters

· (returns)
[out] the X position of the selected camera in meters
· index
[in] the index of the desired camera. the index is zero based

7.2.6.5 RB_CameraYLocation()

The RB_CameraYLocation() function returns the Y position of the selected camera in relation to the coordinate system origin.

float RB_CameraYLocation(int index)

Parameters

· (returns)
[out] the Y position of the selected camera in meters
· index
[in] the index of the desired camera. the index is zero based

7.2.6.6 RB_CameraZLocation()

The RB_CameraZLocation() function returns the Z position of the selected camera in relation to the coordinate system origin.

float RB_CameraZLocation(int index)

Parameters

· (returns)
[out] the Z position of the selected camera in meters

· index
[in] the index of the desired camera. the index is zero based

7.2.6.7 RB_CameraOrientationMatrix()

The RB_CameraOrientationMatrix() function returns the selected element of the orientation matrix of the selected camera.

float RB_CameraOrientationMatrix(int camera, int index)

Parameters

· (returns)
[out] the selected element of the orientation matrix
· camera
[in] the index of the desired camera. the index is zero based
· index
[in] index of the item in the array to retrieve. there are 9 elements in the array, valid inputs range from 0 to 8.

7.2.7 Return Code Processing

7.2.7.1 RB_GetResultString()

The RB_GetResultString() function returns a plain text message for status codes returned from other functions in the Rigid Body API.

const char *RB_GetResultString(NPRESULT result)

Parameters

· (returns)
[out] the plain text message associated with the selected return code
· result
[in] the status code returned from another function in the Rigid Body API

8. Tips and Tricks

8.1 Tracking Environment

The Point Cloud toolkit and Rigid Body toolkit are designed to work in a wide variety of conditions, and the current generation of OptiTrack cameras are more robust and reliable than ever before. There are, however, a number of things that you can do to optimize their

performance :

Distance between the cameras and the user :
The optimum range for tracking markers with OptiTrack cameras is between 0.5 to 6 meters.

Lighting :
We recommend turning off or dimming lights in the room and removing any highly reflective materials that are directly in the view of the OptiTrack cameras. You can easily check what the OptiTrack cameras are seeing by using the Calibration tool Camera Preview step. For details please see Section 4.2.2.

8.2 Tracking Markers

To achieve the best 3D tracking performance NaturalPoint recommends the use of spherical reflective 3D markers which are available from our online store. The markers will be detected most easily when they are clean and there has not been any abrasion to the surface of the marker. If the reflective surface of the markers will come into frequent contact with skin or other surfaces then it is recommended to replace the markers on a regular basis.

9. Cameras and Accessories

NaturalPoint offers a complete line of high quality cameras and accessories to complement your Point Cloud toolkit and Rigid Body toolkit. Please visit our online store to find out more at :
OptiTrack online store catalogue

10 Software Updates

In order to better serve your needs we continually update the Tracking Toolkits software, you may download these updates free of charge from www.naturalpoint.com/optitrack.

To download and install new software, please follow these steps

1. Go to www.naturalpoint.com/optitrack and click on the Support link at the top of the page, then go down to the download section.

2. If the version of software listed for your product is more recent than the version of the software installed on your computer, then download and save the updated installer file to your desktop or temp folder. If a newer version is not available, then there is no need to update your software.

3. Disconnect any OptiTrack cameras from your computer and ensure that no Tracking Toolkits software is running.

4. Use the "Add and Remove Programs" feature of the Windows Control Panel to un-install your existing Tracking Toolkits software.

5. Double click on the downloaded installer .exe file to install the new software. After the program icon reappears on your desktop, you may reconnect OptiTrack cameras to your computer and run the updated software.

11 Troubleshooting

If you are experiencing difficulty using this product, please call 1-541-753-6645 or visit the NaturalPoint website at www.naturalpoint.com, or our online forums at forum.naturalpoint.com for advanced customer support.